DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

LUSTRE USAGE MONITORING

What the &#%@ are users doing with my filesystem?

Kilian CAVALOTTI, Thomas LEIBOVICI | CEA/DAM

LAD'13 – SEPTEMBER 16-17, 2013

LUSTRE ADMINS & DEVELOPERS WORKSHOP 13
PARIS, FRANCE - 2013

MOTIVATION

Lustre monitoring is hard

- very few tools are available
- “distributed filesystem” means “distributed stats”

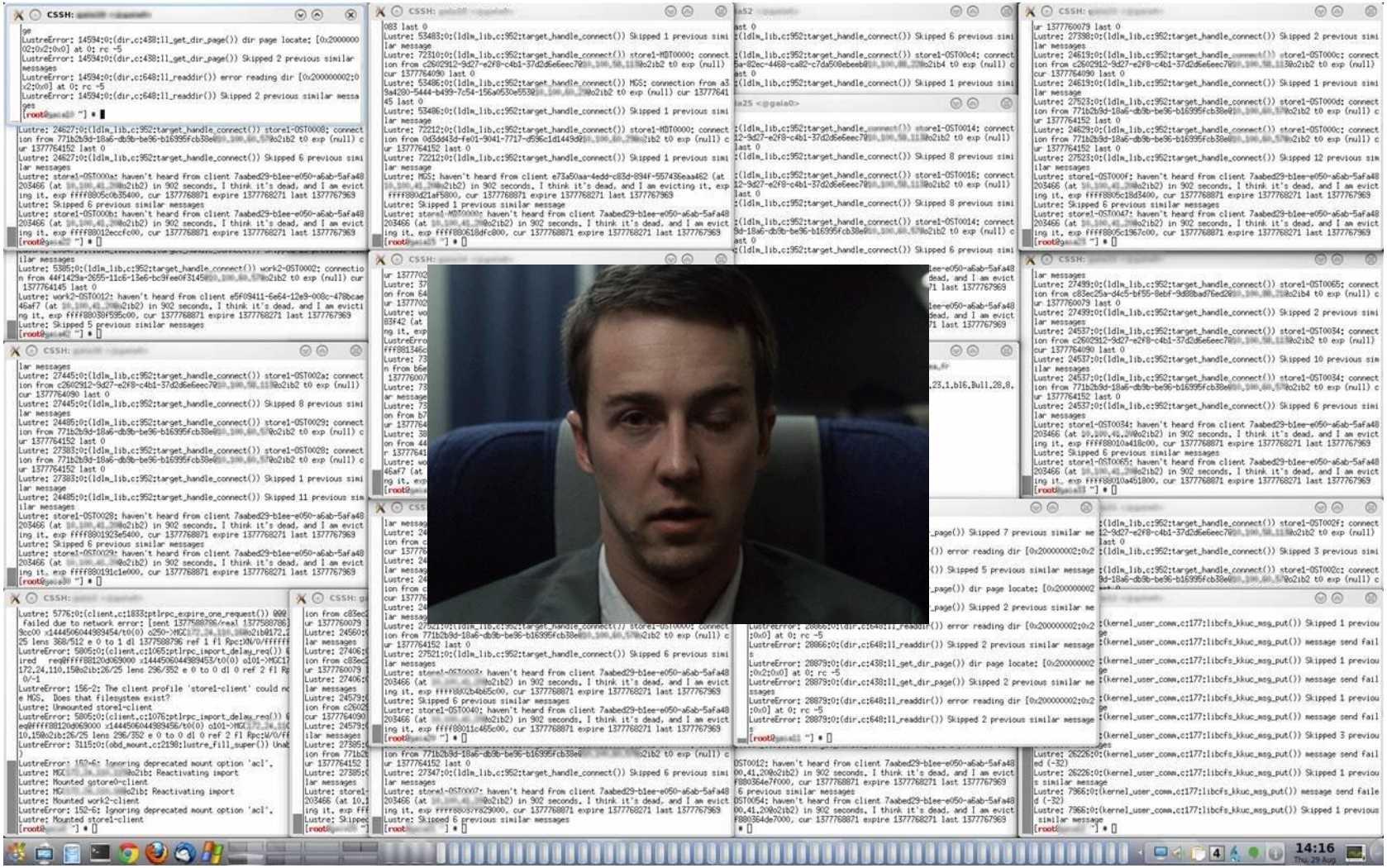
Administrators need stats

- to check that everything runs smoothly
- to analyze problems
- to watch users
  - users are (sometimes unintentionally) malicious, they need to be watched
- because stats are cool
  - and graphs are even cooler

We @CEA/DAM needed tools

- to monitor filesystem activity (performance, trends…)
- to understand filesystem usage (typology, distribution, working sets…)

LUSTRE MONITORING: WHAT WE HAVE



DE LA RECHERCHE À L'INDUSTRIE

cea

LUSTRE MONITORING: WHAT WE WANT

OUTLINE

Monitoring filesystem activity

- tools
- visualizing real-time activity
  - bandwidth, metadata operations, space usage…
- diagnosing slowdowns and unexpected behavior
  - suggesting improvements for user applications causing slowdowns
- understanding how people use their files
  - showing the filesystem "temperature" by visualizing working sets and their evolution

Answering typical questions

- *"I don't get the expected GB/sec..."*
- *"My* ls *takes too long!"*
- *"Can you give me the current usage for each user working on this project?"*
- *"What kind of data is in the filesystem?"*

Disclaimer

- we use Lustre 2.1. (YMMV…)

MONITORING FILESYSTEM ACTIVITY

TOOLBOX

The Lustre admin tool belt

- ltop: github.com/chaos/lmt
  - top-like command providing instant stats about a filesystem
- collectl: collectl.sourceforge.net
  - sar-like tool for the monitoring of Infiniband traffic, Lustre activity, etc.
- htop, dstat and friends

Open-source tools developed @CEA

- shine: lustre-shine.sourceforge.net
  - command-line tool for Lustre filesystem configuration and management
- robinhood policy engine: github.com/cea-hpc/robinhood
  - multi-purpose tool for large filesystem management

Custom scripts

- based on all of the above, glue provided by Bash and Python
  - we parse `/proc/fs/lustre` a lot
- nice colors courtesy of RRDtool

ROBINHOOD POLICY ENGINE

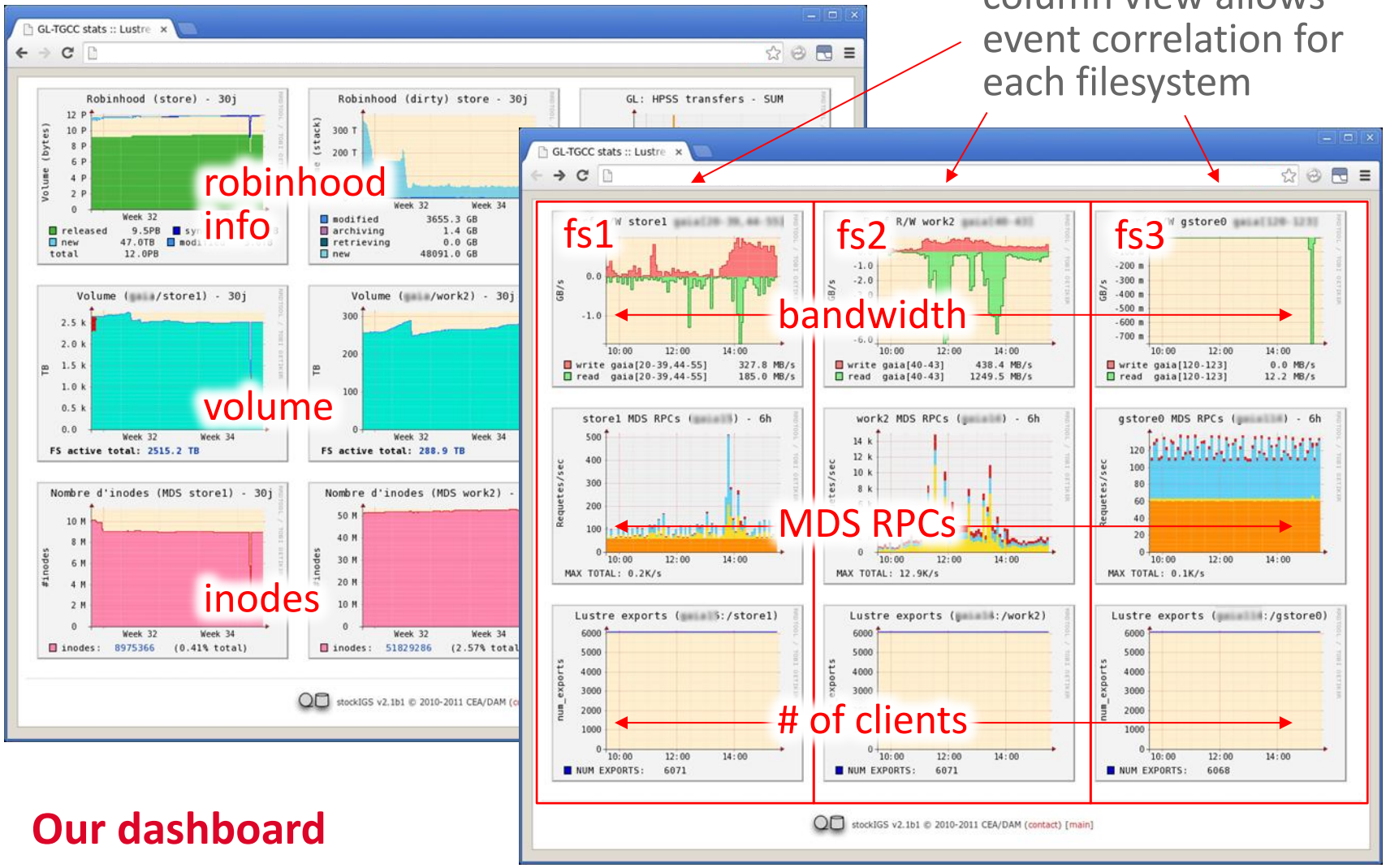
Swiss-army knife for your filesystems

- audit, accounting, alerts, migration, purge, based on policies
- massively parallel
- stores filesystem metadata in a (My)SQL database
  - allows lightning-fast usage reports
  - provides `rbh-du` and `rbh-find` replacements for `du` and `find`
- advanced capabilities for Lustre filesystems
  - purge on OST usage, list files per OST
  - pool and/or OST based policies
  - changelogs reader (Lustre v2): no scan required
- reporting tools (CLI), web interface (GUI)
- open-source license
- http://robinhood.sourceforge.net

@CEA

- one instance per Lustre filesystem, using changelogs

FILESYSTEM ACTIVITY OVERVIEW



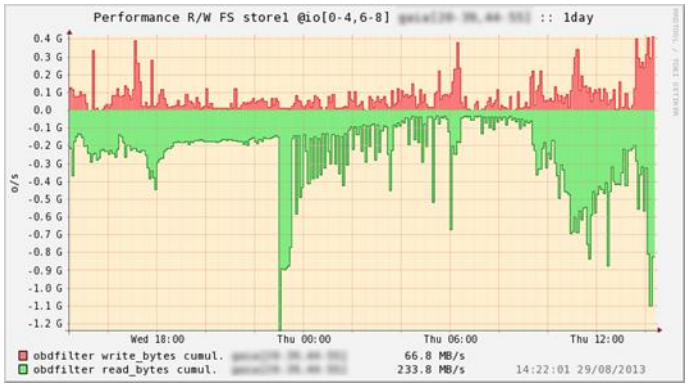
Our dashboard

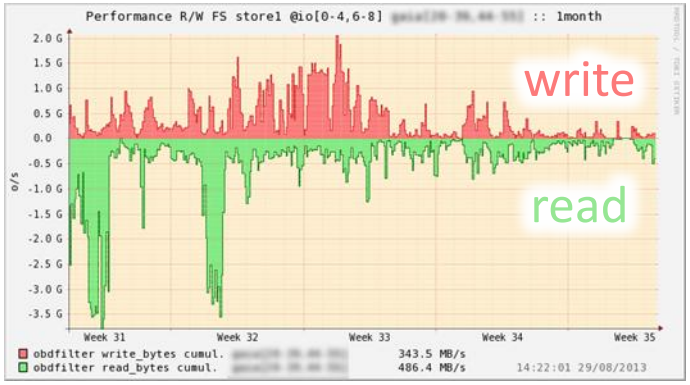
FILESYSTEM ACTIVITY: BANDWIDTH

Bandwidth

- /proc/fs/lustre/obdfilter/${fsname}-OST*/stats

```
snapshot_time             1377780180.134516 secs.usecs
read_bytes                178268 samples [bytes] 0 1048576 185094526864
write_bytes               220770 samples [bytes] 115 1048576 230859704715
get_page                  399038 samples [usec] 0 6976 69863795 15160124841
cache_access              45189234 samples [pages] 1 1 45189234
cache_miss                45189234 samples [pages] 1 1 45189234
get_info                  5250 samples [reqs]
set_info_async            25 samples [reqs]
process_config            1 samples [reqs]
connect                   6074 samples [reqs]
disconnect                12 samples [reqs]
statfs                    181816 samples [reqs]
create                    30 samples [reqs]
destroy                   520 samples [reqs]
setattr                   76 samples [reqs]
punch                     19 samples [reqs]
sync                      1092 samples [reqs]
preprw                    399038 samples [reqs]
commitrw                  399038 samples [reqs]
llog_init                 2 samples [reqs]
quotacheck                1 samples [reqs]
quotactl                  1060 samples [reqs]
ping                      10349451 samples [reqs]
```

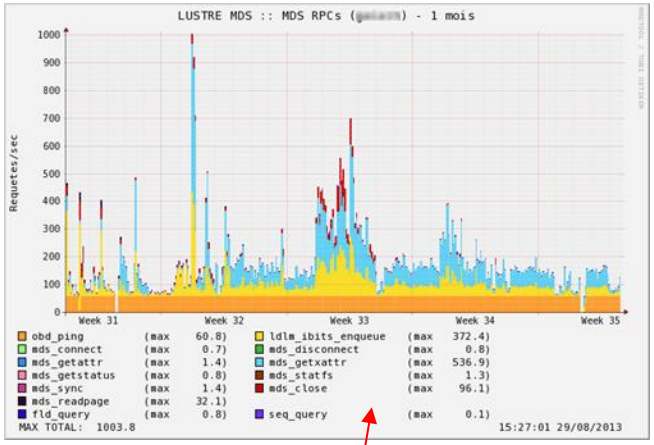




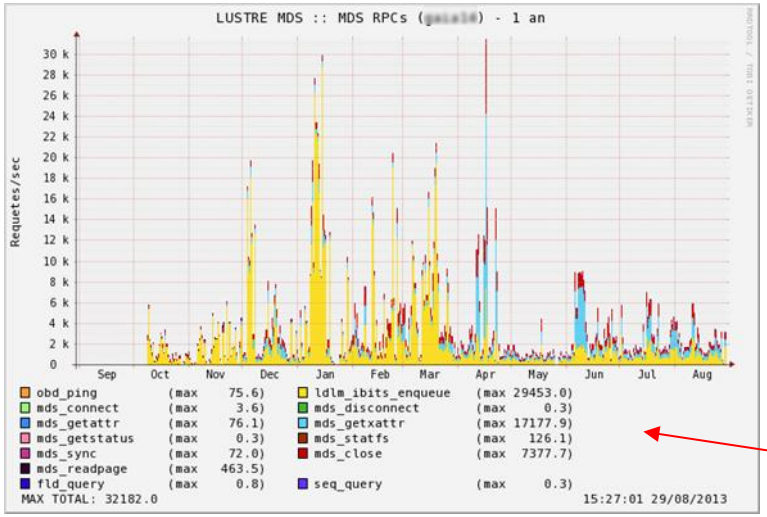
- aggregated on each OST, for each filesystem

Metadata operations (MDS RPCs)

- `/proc/fs/lustre/mdt/${fsname}-MDT0000/mdt*/stats`
- most commonly used:
  - `obd_ping`: filesystem health
  - `mds_(dis)connect`: clients (u)mounts
  - `mds_get(x)attr`: metadata operations
  - `mds_statfs`: (lfs) df
  - `mds_readpage`: readdir()
  - `ldlm_ibits_enqueue/mds_close`: ~open()/close()



long-term storage filesystem: large files, few metadata operations



2 filesystems with the same clients (~6000), but different usages

scratch-like filesystem: lots of open()

DIAGNOSING SLOWDOWNS (1/2)

From graph to clients

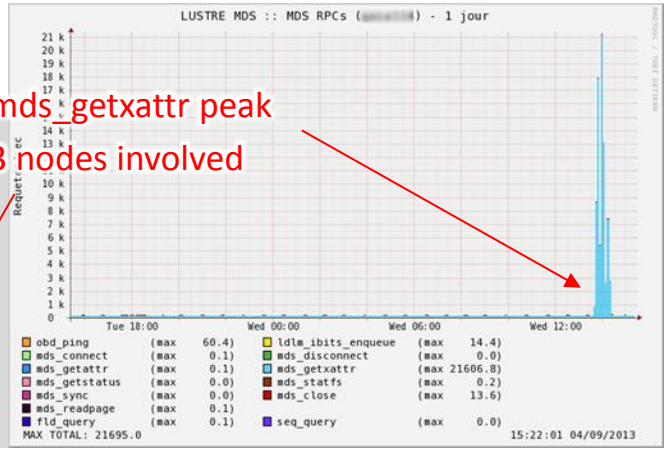
- peak detection on the RPC graph
  - initially a manual process: watching the graph
  - now automatic: averaging the last 10 values in the RRD DB, and comparing to a threshold
- collection of RPC samples
  - initially a manual process: enabling debug traces, collecting, disabling debug traces
  - now automatic: background collection for 5sec every 5min, keeping 7 days of traces

```
echo +rpctrace > /proc/sys/lnet/debug
echo +rpc > /proc/sys/lnet/subsystem_debug
```

- analysis of collected samples
  - RPC type breakdown by Lustre client

```
GENERAL
Source nodes:             node114  (1)
Duration:                 5.0 sec
Total RPC:                  88105  (17618.5 RPC/sec)
Ignored ping RPC:             102

NODENAME       NET    COUNT       %  CUMUL   DETAILS
--------       ---    -----       -  -----   -------
10.100.10.137 o2ib6 18567   21.07  21.07   mds_getxattr: 18567
10.100.10.138 o2ib6 17534   19.90  40.97   mds_getxattr: 17534
10.100.10.140 o2ib6 14724   16.71  57.69   mds_getxattr: 14724
```



DIAGNOSING SLOWDOWNS (2/2)

From client identification to code improvement

- querying the cluster scheduler

```
$ sacct -S startdate -E endate -N nodename
```

  getting jobs running on those nodes during the event

  contacting the user

- for live events, possibility to `strace` the process at fault

  having seen such things as:

```
# strace -e stat -p 95095 -r
Process 95095 attached - interrupt to quit
     0.000000 stat("/path/to/case/WORKDIR/DMP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.000398 stat("/path/to/case/WORKDIR/BF", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.000246 stat("/path/to/case/WORKDIR/STOP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.038258 stat("/path/to/case/WORKDIR/DMP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.000338 stat("/path/to/case/WORKDIR/BF", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.000275 stat("/path/to/case/WORKDIR/STOP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.038236 stat("/path/to/case/WORKDIR/DMP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.000320 stat("/path/to/case/WORKDIR/BF", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
     0.000234 stat("/path/to/case/WORKDIR/STOP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
```

  kindly suggested the user better ways to stop his run

FILESYSTEM TEMPERATURE: USER WORKING SETS

Time-lapse of filesystem usage

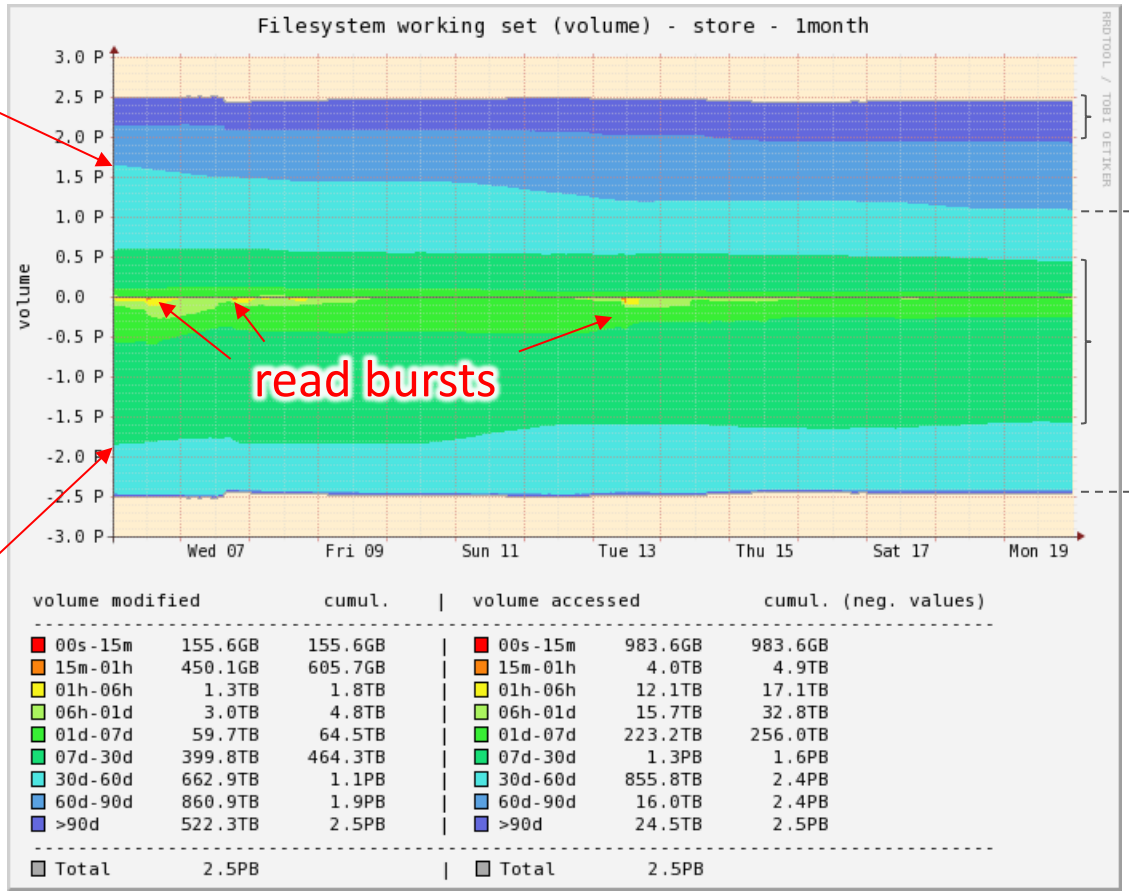
- working set = set of files *recently* written/read

70% of data produced within the last 2 months

data **production** (mod. time)

data **in use** (last access)

80% of data accessed <1 month



"Cold" data

1 month working set

2 months working set

FILESYSTEM TEMPERATURE: DETAILS

Implementation

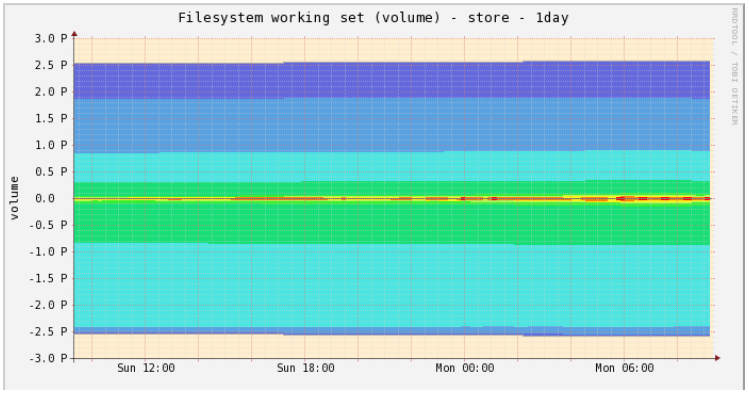
- compute {modification,access} age for each file
  - `now - last_mod/last_access`
- aggregate count/volume by age range
- based on Robinhood database
  - optimized SQL query
  - could be slow for filesystems with > 100s millions of inodes
- some RRDtool magic involved
  - the RRDtool graph command line is ~6500 characters long…

Benefits

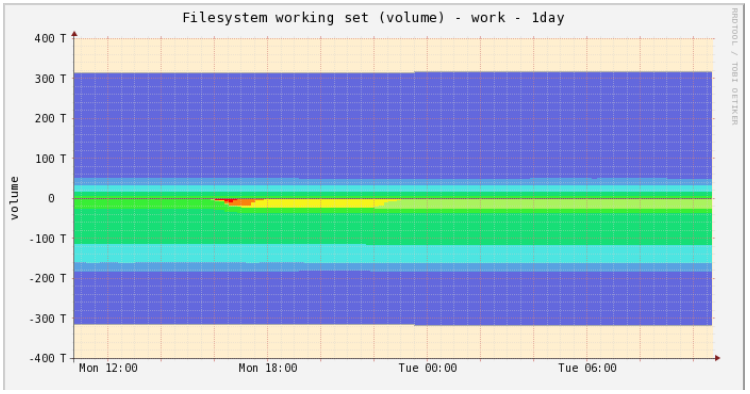
- visualization of the current working set
  - age repartition, based on last modification and last access dates
- allows to anticipate filesystem evolutions
  - mostly recently written data: need to add more OSTs
  - mostly old, never-accessed files: could be purged

FILESYSTEM TEMPERATURE: EXAMPLES

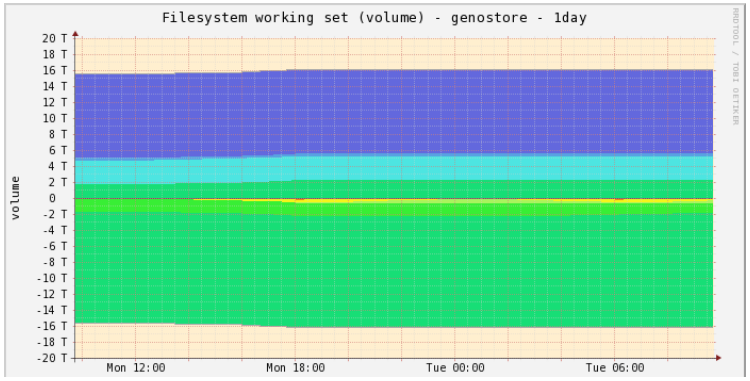
Visualization of different filesystem usage patterns



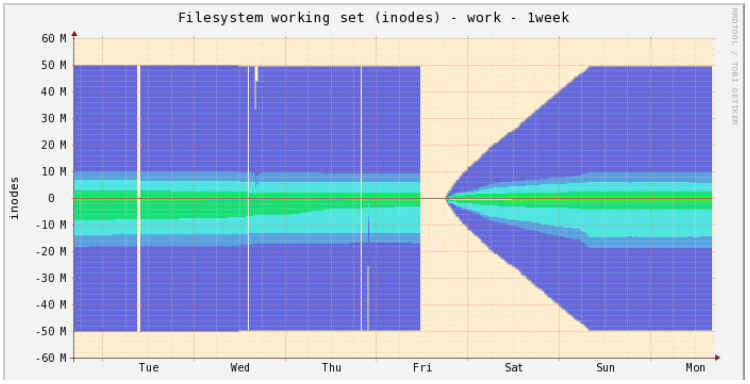
significant reads/writes (volume)



cooling effect after a large read (volume)



read-mostly filesystem (volume)



Robinhood DB dump and initial scan (inodes)

nice linear scan, ~2 days for 50M inodes

ANSWERING TYPICAL QUESTIONS

a.k.a.

THE ROBINHOOD COOKBOOK

ENFORCING FILESYSTEM RECOMMENDED USAGE

Question

- *"I don't get the expected GB/sec..."*

Different filesystems, different usages

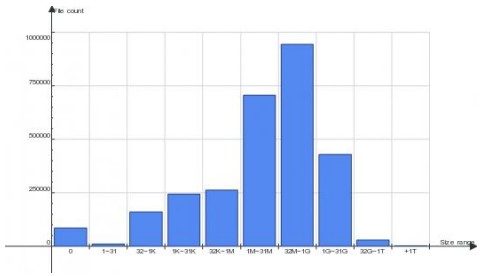
- user documentation indicates recommended file sizes for each filesystem
  - as well as purge policies, quotas…
- often, recommendations not followed → bad performance
- usually a case of "small files"
  - writing large blocks is more efficient
  - metadata overhead decreases performance

Implemented solutions

- filesystem quotas
  - a limited amount of inodes motivates users to write bigger files
- user guidance
  - file size profiling, small file detection using Robinhood
  - information provided to users if necessary, helping them to take appropriate measures

File size profiling

- available in the Robinhood web interface
- file size repartition
  - global / per user



- summary per user

User	Total volume	File count	Avg file size	file size ratio			
				empty	<1K	<1M	<1G
	11.98 TB	36 251	346.48 MB	0.74%	3.17%	67.12%	97.65%
	15.02 TB	86 584	181.94 MB	21.73%	36.00%	86.33%	93.63%
	120.02 TB	114 445	1.07 GB	13.30%	30.75%	65.83%	75.80%
	27.31 TB	65 944	434.31 MB	21.16%	30.35%	75.72%	84.30%
	7.53 TB	27 725	284.86 MB	0.76%	26.44%	80.27%	90.80%
	19.17 TB	87 617	229.41 MB	9.19%	22.32%	61.84%	81.17%
	74.17 TB	38 685	1.96 GB	16.62%	22.65%	50.07%	60.83%
	57.35 TB	112 730	533.41 MB	1.97%	6.21%	36.07%	65.95%

User guidance

- informational emails sent on specific criteria
  - avg. file size < 50MB, more than 80% of files < 32MB
  - includes good I/O practices, stats about their account (list of non-compliant directories)
- graduated response system
  - first two emails are warnings
  - new job submission is suspended after the 3rd warning
  - without any action or feedback from the user, data is automatically packed

INFORMATIONAL EMAILS: IMPLEMENTATION

Robinhood reports

- file size profiling

  get a list of users sorted by the ratio of files in a given size range

```
$ rbh-report --top-users --by-szratio=1..31M

rank, user    , spc_used,  count,  avg_size, ratio(1..31M)
1, john       ,  1.23 TB,  26763,  60.59 MB,          91.30%
2, perrez     , 80.77 GB,  27133, 114.71 MB,          87.64%
3, matthiew   ,  2.22 TB,  25556,   1.04 GB,          85.76%
4, vladimir   ,    62 GB,  14846,  74.18 MB,          78.50%
5, gino       , 30.92 GB,  15615,  77.34 MB,          77.02%
```

  tip: with "`-l FULL`", Robinhood logs SQL requests, for reuse and customization

- small file detection

  list directories with at least 500 entries, sorted by average file size (smallest first)

```
$ rbh-report --top-dirs --by-avgsize --reverse --count-min=500
```

LARGE DIRECTORY DETECTION

Question

- *"My* ls *takes too long!"*

Issue

- directories with too many entries (100,000s)
- sequential processing of `readdir()` takes a long time

Implemented solutions

- real-time directory alerts

  alerts defined in Robinhood configuration

```
Alert large_dir { type == directory
                  and dircount > 10000
                  and last_mod < 1d }
```

  ⚠ doesn't work with changelogs, only when scanning

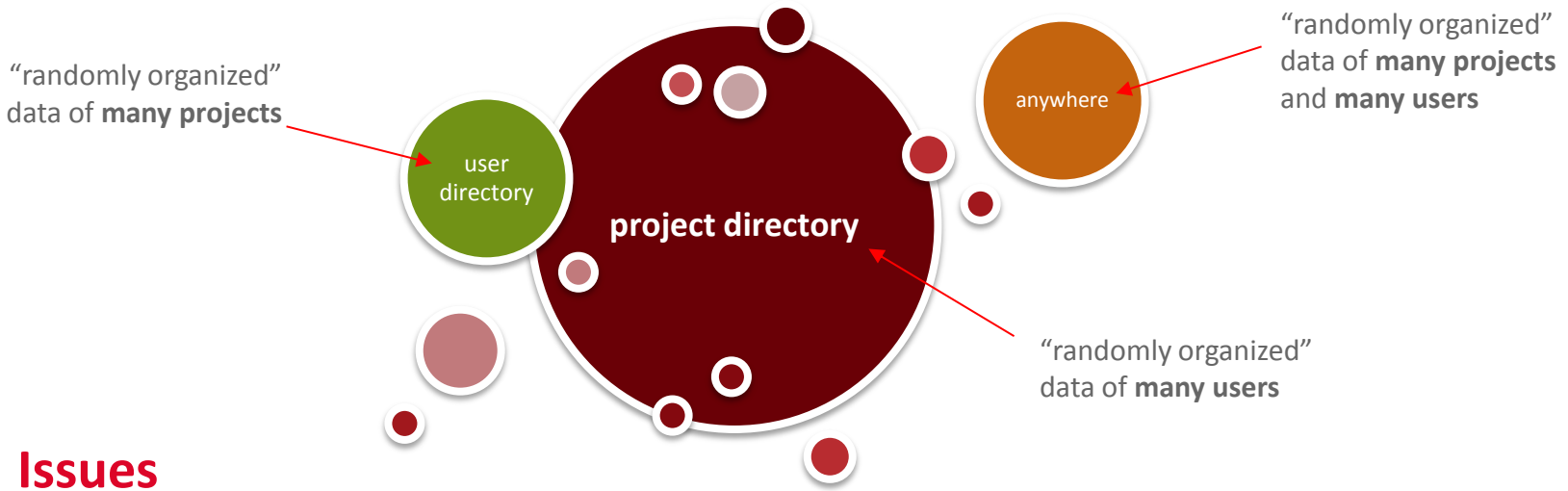
- report directories with the most entries

```
$ rbh-report --topdirs
```

ACCOUNTING (1/2)

Question

- *"Can you give me the current usage for each user working on this project?"*
- given that:
  - some data are in a project directory,
  - users have some other data from the same project in their own directory,
  - project data can sometimes be identified by a group attribute, and be anywhere else ...



Issues

- data belonging to a single project can be distributed in several locations
- logical attachment to a project can reside in file attributes, not in location

ACCOUNTING (2/2)

Implemented solution

- basic accounting using `rbh-report`
- space used in the whole filesystem
  - by user (splitted by group)

```
$ rbh-report -u foo* -S

user ,     group,     type,      count,   spc_used,   avg_size
foo1 ,   proj001,     file,     422367,   71.01 GB,  335.54 KB
…
Total: 498230 entries, 77918785024 bytes used (72.57 GB)
```

  - for a specific group

```
$ rbh-report -g proj001
```

- per directory accounting using `rbh-du` filtering capabilities
  - by a given user in a given directory

```
$ rbh-du -u theuser /project_dir
```

  - by a given group in given directories

```
$ rbh-du -g proj001 /users/foo*
```

- project data in other locations

```
$ rbh-find -g proj001
```

CONTENT MONITORING

Question

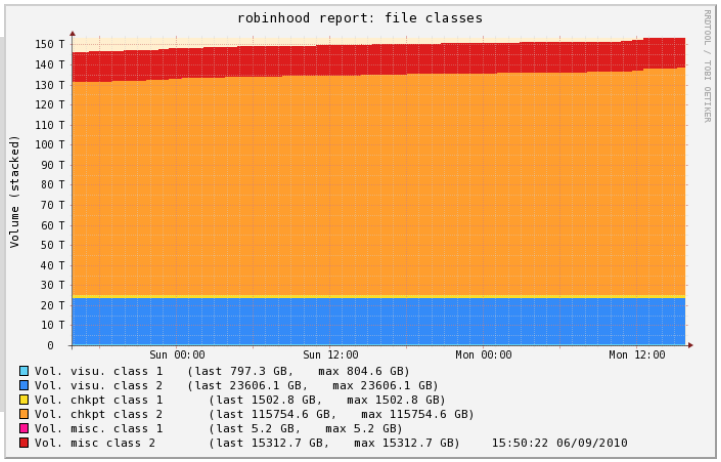
- *“What kind of data is in my filesystem?”*

Implemented solution

- define Robinhood file classes
- arbitrary definitions based on file attributes

  name, owner, group, size, extended attributes…

```
Fileclass system_log_files {
    definition { name == “*.log” and
                 (owner == “root” or group == “root”) }
}
Fileclass big_pictures {
    definition { (name == “*.png” or name == “*.jpg”)
                   and (size > 10MB) }
}
Fileclass flagged {
    definition { xattr.user.flag == “expected value” }
}
```



- get class summary

```
$ rbh-report --classinfo
class             ,    count,   spc_used,     volume,  min_size,    max_size,    avg_size
Documents         ,   128965,  227.35 TB,  250.69 TB,   8.00 KB,     2.00 TB,    30.03 GB
System_log_files  ,     1536,    4.06 TB,    4.06 TB,        684,  200.01 GB,     2.71 GB
Big_pictures      ,   621623,  637.99 TB,  638.02 TB,          3,     1.54 TB,     1.05 GB
```

WRAPPING UP

CONCLUSION

Visualizing filesystem activity is useful

- we can diagnose abnormal patterns
  - and improve suboptimal code
- we can better understand user behavior
  - and help them getting the best I/O performance out of their applications

Solutions exist

- some assembly required
- but all the required information is there
  - in `/proc/fs/lustre`
  - in the filesystem metadata (Robinhood database)
- you need Robinhood
  - and some RRDtool-fu

THANK YOU!

QUESTIONS?

Commissariat à l'énergie atomique et aux énergies alternatives
CEA / DAM Ile-de-France | Bruyères-le-Châtel - 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00

Direction des applications militaires
Département sciences de la simulation et de l'information
Service informatique scientifique et réseaux